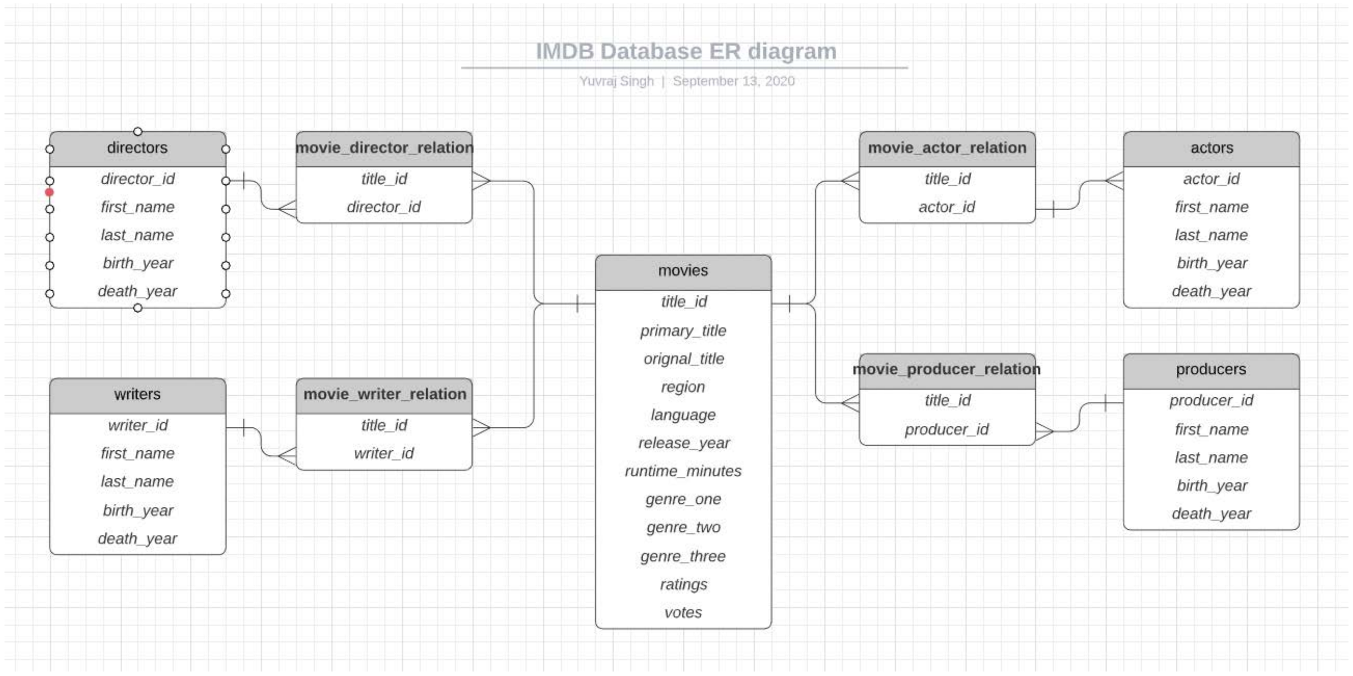
CSCI 620 – Intro to Big Data
Assignment #1 – Relational Model

Name: *Yuvraj Singh*

1) ER Diagram



2) Relational Model Script

See attached .sql file for information about the database schema. To run the sql file, one could either use PGAdmin, or use PSQL to connect to the database and run the create_tables.sql file. What I did was use Windows Powershell and ran following command to create the tables, ".\psql.exe -U postgres -d imdb -f C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment1\create_tables.sql", With this command, you have to specify the postgres user, the database, and the directory of the .sql file. As for sql file itself, I ran into a few things when trying to create it such as deciding the data types for each attribute and how to handle the genre entries for each movie. Since not all movies had three genres, I decided to parse the genre column of the tsv file and split it up into three genre attributes in the movies table. I may have to look into a different way to store genres since having some rows with empty data for the genre #2 and genre #3 columns is not quite efficient. As for the datatypes, I tried to limit the attributes to what I thought the maximum data that would be stored for each attribute.

3) IMDB Dataset Description

title.basics.tsv: This dataset contains most of the information for each title, type (like movie, tv episodes), start/end year (release year for movies), duration, and genres. Some of the titles had multiple genres. This file is used to create the initial movie entries in the movie table, but it is still lacking some information. Not all the data was used in this dataset because some of it was irrelevant due to the assignment scope such as the isAdult attribute.

title.akas.tsv: This dataset contains additional information about each movie like the region, and language they are available in, if it's an original title, or if it's a DVD, TV, video, etc… I found most of information in this dataset not important, so it was not used to provided additional information of each movie.

title.crew.tsv: This dataset contains information about the directors and writers for each specific title. To be specific, the contents tied directors and writers to each specific title, allowing us to form the relation between movie-director and movie-writer.

title.episode.tsv: This dataset contains information for each tv series, like the season and episode numbers and which title they belong to. This file was disregarded since the scope of the assignment was limited to movies only.

title.principals.tsv: This dataset contains information for each title. It essentially tied some person to a title, and it provided information as to their job for that movie. With the contents of this file, the relationship between a title and a specific person can be formed. This file is very similar to the **title.crew.tsv** file.

title.ratings.tsv: This dataset contains the ratings and number of votes for a specific title. With this, additional information can be assigned for each title in the movies table.

names.basics.tsv: This dataset contains information about people, like their ID, name, date of birth, decessed year, primary job, etc…. The contents of this file is used to populate the actors, directors, writers, and producers' tables with initial information.

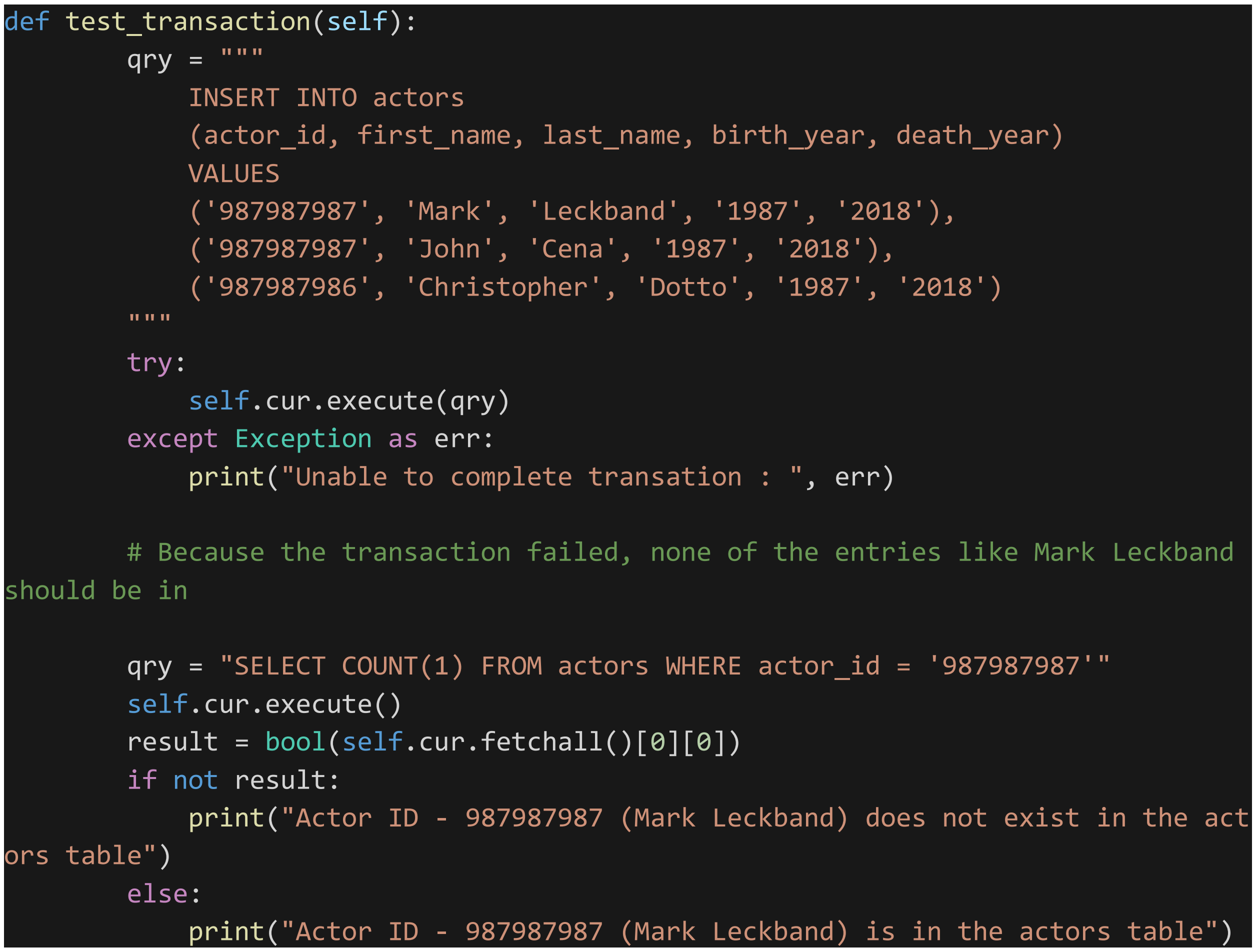
4) Inserting Data to Database

See attached python file for more information. To be able to run the IMDB.py file, the psycopg2 library will need to be installed. In the IMDB.py file, the configuration to connect to the postgres database needs to be entered. The default postgres database configuration is already entered. For the file to run properly, each folder for the datasets needs to be in the same directory as the IMDB.py file.

In the IMDB.py file, I choose which datasets I deemed valuable for the assignment scope and I created separate methods to open them and to parse and clean up the data for each row in the tsv files. After the data has been obtained and formatted, it was then passed into another method that handled the Postgres commands, such as either inserting the data into its respective table or checking if certain things existed. The biggest thing I noticed when creating this program to load the IMDB data into the database was timing concerns. The way I had done the SQL insertions was by passing in a dictionary of values to a method that would execute the insertion along with the parameters, thus doing the insertions one-by-one. By doing this, I noticed that some methods took quite a while to finish, like reading the names.basic.tsv file and creating each person in their respective table took approximately 2349 seconds (~ 77 mins) alone. Inserting each movie and updating it took approximately 500 seconds (~ 8 mins). Creating the relations for the actors/directors/writers/producers to movies took the longest, with approximately 6671 seconds (~ 111 mins). The timing of program is really close to the proposed 4-hour limit so in the future, I will have to find some way to speed up the SQL queries. I did find some ways to speed this up but did not implement such because I wanted to research it a bit more. One way that I could have made this faster is by doing a multirow insert statements instead of multiple insert statements, but one issue I thought about is that if one of the inserts fails, the whole transaction would be aborted. I noticed a lot of Duplicate Key exceptions are being thrown when trying to insert the relations between crew to movie so I assumed the whole transaction would be aborted. This is unfortunate since this function is where a lot of time is being wasted. Another method in speeding this up would be using PostgreSQL Upsert, which is something I would like to investigate more before trying to implement this.

5) Transaction Error Check

The issue with inserting many rows in one insert statement is that if a violation occurs, the whole transaction is aborted. For example, in the code below, three actors are to be inserted, while the second one has the same actor_id as the first row. Because of this, a duplicate primary key violation is thrown and the whole transaction is aborted. To verify that the transaction is aborted, we query the actors table to see if we can find the actor_id of "Mark Leckband", if so, result will be True, if not result will be False.

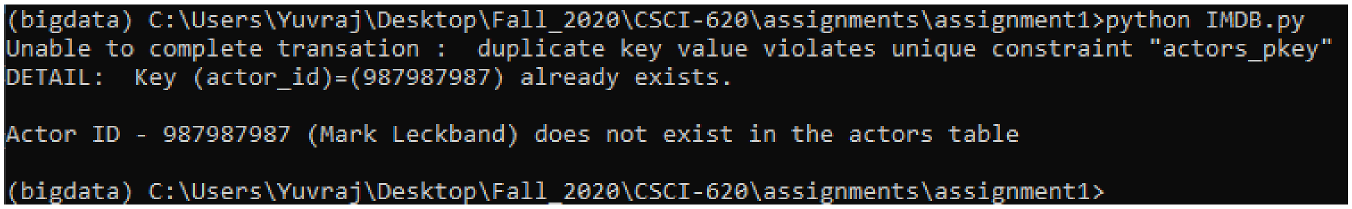


```
def test_transaction(self):
        qry = """
            INSERT INTO actors
            (actor_id, first_name, last_name, birth_year, death_year)
            VALUES
            ('987987987', 'Mark', 'Leckband', '1987', '2018'),
            ('987987987', 'John', 'Cena', '1987', '2018'),
            ('987987986', 'Christopher', 'Dotto', '1987', '2018')
        """
        try:
            self.cur.execute(qry)
        except Exception as err:
            print("Unable to complete transation : ", err)

        # Because the transaction failed, none of the entries like Mark Leckband
should be in

        qry = "SELECT COUNT(1) FROM actors WHERE actor_id = '987987987'"
        self.cur.execute()
        result = bool(self.cur.fetchall()[0][0])
        if not result:
            print("Actor ID - 987987987 (Mark Leckband) does not exist in the act
ors table")
        else:
            print("Actor ID - 987987987 (Mark Leckband) is in the actors table")
```

When running the code, we can see the output message below, showing that the actor_id for "Mark Leckband" does in fact not exist, therefore proving that the whole transaction is aborted due to the violation.



```
(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment1>python IMDB.py
Unable to complete transation :  duplicate key value violates unique constraint "actors_pkey"
DETAIL:  Key (actor_id)=(987987987) already exists.

Actor ID - 987987987 (Mark Leckband) does not exist in the actors table

(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment1>
```